Programmation fonctionnelle
TP n° 5

Fractale de Pythagore

On cherche à afficher des fractales de Pythagore. Ces fractales se construisent par génération successive à partir de segments actifs. Si un segment AB est actif à la génération n, on construit la carré $ABCD$ puis le triangle rectangle CDE. Le procédé est illustré ci-dessous (Figure 1), l'angle θ est un paramètre. Les segments DE et EC deviennent actifs pour la génération suivante.

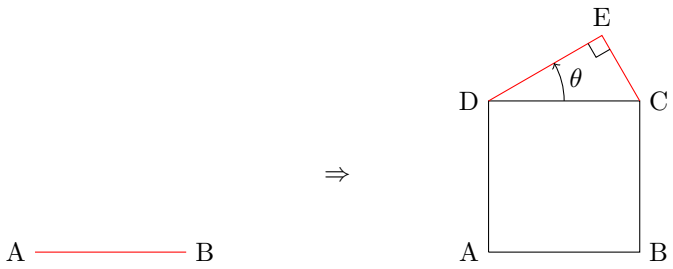


FIGURE 1 – Principe de construction

La génération 0 est composée d'un unique segment horizontal actif. La fractale de Pythagore est la limite de la figure obtenue après un nombre infini de générations. Des exemples de générations sont donnés (Figure 2).

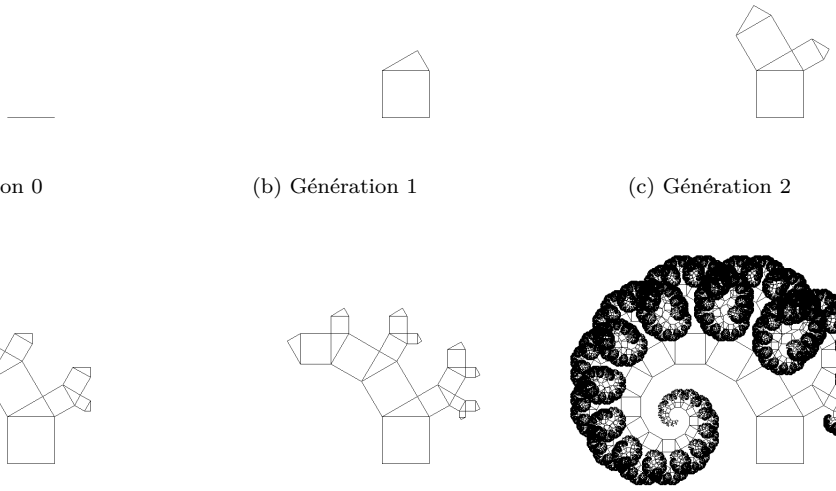
Comme il va être nécessaire de dessiner, faites commencer votre fichier par les lignes :

```
#load "graphics.cma";;
Graphics.open_graph " 900x600";;
```

Pour construire une figure comme la fractale de Pythagore, le plus simple est de passer par de la géométrie vectorielle. Pour cela, on se donne les types suivants.

```
type point = {abs : float; ord : float}
type vector = {x : float; y : float}
type segment = {p1 : point; p2 : point}
```

1. Définissez une fonction `make_vector : point -> point -> vector` qui construit le vecteur $\overrightarrow{AB}$ à partir des points A et B.
2. Définissez une fonction `mul_scal : float -> vector -> vector` qui multiplie un vecteur par un scalaire.
3. Définissez une fonction `rotate_vector : vector -> float -> vector` qui renvoie l'image d'un vecteur par une rotation d'angle θ. Pour rappel, si $\vec{u}$ est un vecteur de coordonnées (x, y), son image par la rotation d'angle θ est le vecteur $\vec{v}$ de coordonnées $(x\cos\theta - y\sin\theta, x\sin\theta + y\cos\theta)$.



(a) Génération 0 (b) Génération 1 (c) Génération 2

(d) Génération 3 (e) Génération 4 (f) Génération 40

FIGURE 2 – Exemple de génération pour $\theta = \pi/6$.

4. Définissez une fonction `translate_point : point -> vector -> point` qui renvoie l'image d'un point par une translation de vecteur $\vec{u}$.
5. Définissez une fonction `draw_segment : segment -> unit` qui dessine un segment dans la fenêtre graphique.
6. Définissez une fonction `process_segment : segment -> float -> segment * segment` qui effectue une étape du calcul pour un segment. L'appel `process_segment ab theta` affiche le carré et le triangle rectangle produit à partir du segment `ab` et renvoie le couple formé par les segments `de` et `ec`.
7. Définissez une fonction `pythagore : int -> float -> segment -> unit` qui applique la fonction précédente à une liste de segments et qui itère le processus. L'appel `pythagore n theta ab` calcule la `n`$^{\text{e}}$ génération à partir du segment `ab`.

Fractale de Mandelbrot

.

L'ensemble de Mandelbrot est une figure fractale définie de la manière suivante. Un point M d'affixe c appartient à l'ensemble de Mandelbrot si et seulement si la suite z telle que

$$\begin{cases} z_0 = 0 \\ z_{n+1} = z_n^2 + c \end{cases}$$

est bornée.

La figure obtenue est particulièrement complexe. On en obtient des visualisations par des approximations numériques. On sait que si un élément z_k de la suite vérifie $|z_k| > 2$ alors le point M correspondant n'appartient pas à l'ensemble. En itérant le calcul des valeurs de la suite, on peut alors déterminer des frontières de plus en plus précises de la figure.

On représente un pixel du plan par le type `pixel`.

```
type pixel = {
  coord : int * int; (* cooordonnées graphiques *)
  affix : Complex.t; (* affixe pour la position logique *)
  mutable value : Complex.t; (* valeur courante de la suite z *)
  mutable out : bool; (* indique si |value| > 2 *)
}
```

1. Définissez une fonction `init_pixel i j x y d` qui crée un pixel de coordonnées graphiques `(i, j)`. Les paramètres `x` et `y` correspondent aux coordonnées logiques du points de coordonnées graphiques `(0, 0)` (son affixe est donc $\texttt{x} + i\texttt{y}$). Le paramètre `d` correspond à la distance logique entre deux pixels adjacents.
2. Définissez une fonction `make_pixels m n x y d` qui crée une matrice de pixels de taille $\texttt{m} \times \texttt{n}$. Les autres paramètres ont le même sens que dans la question précédente.
3. Définissez une fonction `compute_next pixel` qui met à jour le pixel passée en paramètre. Si on sait déjà qu'il n'appartient pas à la fractale, on l'ignore. Dans le cas contraire, on calcule la valeur suivante de la suite z et on la stocke. Si elle permet de déterminer que le pixel n'appartient pas à l'ensemble, on l'affiche dans la couleur courante (utilisez la fonction `Graphics.plot i j`).
4. Définisse une fonction `get_value array k` qui renvoie le $\texttt{k}^{\text{e}}$ élément du tableau `array` modulo sa taille.
5. Définissez une fonction `mandelbrot w h n colors` qui affiche la fractale de Mandelbrot après `n` itérations de la suite z. L'image produite est de taille `w` par `h`. Le tableau `colors` donne la couleur dans laquelle on affiche les pixels qui sortent de l'ensemble. À chaque itération, on avance d'une couleur dans le tableau.

   Pour obtenir un affichage correct, le pixel de coordonnées graphiques $(0, 0)$ a les coordonnées logiques $(-2, -1)$ et le pixel de coordonnées graphiques $(w - 1, h - 1)$ a pour coordonnées logiques $(1, 1)$. Il faut par ailleur supposer que le rapport w/h soit égal à $3/2$ (ou alors faites un petit calcul pour régler le problème).

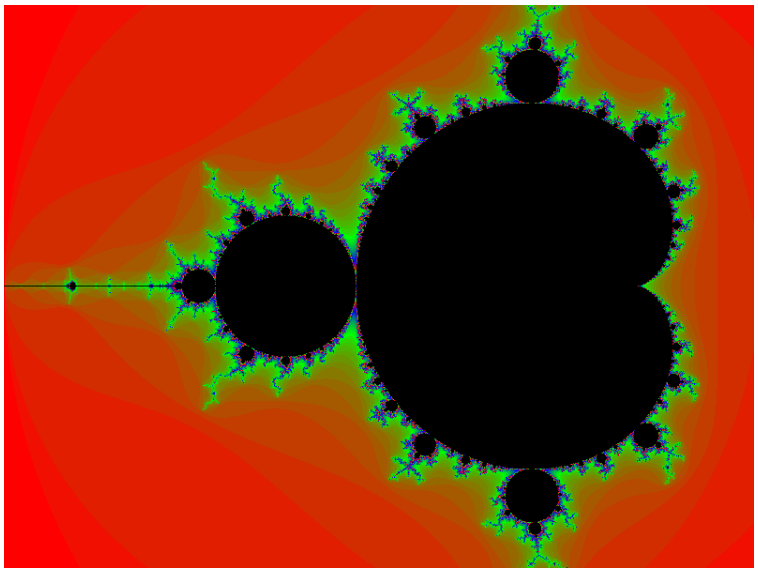


FIGURE 3 – La fractale de Mandelbrot